CSCE475/875 Multiagent Systems

Handout 4: Notes on Distributed Optimization

January 21, 2020

Based on Shoham and Leyton-Brown (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*, Cambridge.

1. Introduction

How can agents optimize a global objective function? Specifically, we consider four families of techniques and associated sample problems:

- Distributed dynamic programming (as applied to path-planning problems);
- Distributed solutions to Markov Decision Problems (MDPs);
- Optimization algorithms with an economic flavor (as applied to matching and scheduling problems) (auctions and contract nets);
- Coordination via social laws and conventions, including voting.

2.1. Distributed Dynamic Programming for Path Planning

2.1.1. Asynchronous Dynamic Programming

Underlying our solutions is the **principle of optimality**: if node *x* lies on a shortest path from *s* to *t*, then the portion of the path from *s* to *x* (or, respectively, from *r* to *t*) must also be the shortest paths between *s* and *x* (resp., *x* and *t*). This allows an incremental divide-and-conquer procedure, also known as *dynamic programming*. ***Notes*:** It is **complete**, **optimal**, but **not scalable**.

- The shortest distance from any node *i* to the goal *g* as $h^*(i)$.
- The cost for the link between nodes *i* and *j* is $w(i,j)$.
- The shortest distance from *i* to the goal *g* via a node *j* neighboring *i* is:

$$f^*(i,j) = w(i,j) + h^*(j)$$

- $h^*(i) = \min_j f^*(i,j)$ (by the principle of optimality)

Procedure ASYNCHDP(node *i*)
if *i is a goal node* **then**
  $h(i) \leftarrow 0$
else
  initialize $h(i)$ arbitrarily (e.g., to infinity or 0)
endif
repeat
  forall neighbors *j* **do**
    $f(i,j) = w(i,j) + h(j)$
  end for
  $h(i) \leftarrow \min_j f(i,j)$
end repeat
End Procedure

2.1.2. Learning Real-Time A* (LTRA*)

Here, the agent starts at a given node, performs an operation similar to that of asynchronous dynamic programming, and then *moves* to the neighboring node with the shortest estimated distance to the goal,

and repeats. This is particular useful when there is one agent, and also there is advantage to *interleave planning and execution*.

Procedure LRTA*
$i \leftarrow s$ *// the start node*
while *i* is **not** a goal node **do**
  foreach neighbor *j* **do**
    $f(j) = w(i,j) + h(j)$
  end for
  $i' \leftarrow arg\min_j f(j)$ *// breaking ties at random*
  $h(i) \leftarrow max\big(h(i), f(i')\big)$ *// admissibility*
  $i \leftarrow i'$ *// move to the neighbor*
end while
End Procedure

Notes:

- *h* must be **admissible**: *h* never overestimates the distance to the goal, i.e.,. $h(i) \leq h^*(i)$. (WHY?)
- **Complete. Optimal** given enough trials.
- Multiple agents? (1) agents have different ways of breaking ties, and (2) all have access to a shared *h*-value table.

2.2. Action Selection in Multiagent MDPs

Background. A Markov Decision Process (MDP) is a discrete time *stochastic* (non-deterministic!) control process. At each time step, the process is in some state *s*, and the decision maker may choose any action *a* that is available in state *s*. The process responds at the next time step by *probabilistically* transitioning into a new state s' as a result of performing *a*. This transition or step gives the decision maker a reward for that state-action decision: $r(s,a,s')$.

To help a decision maker make better decisions, one will need to know the probability that the process moves into its new state s' as influenced by the chosen action. Specifically, it is given by the state transition function $p(s,a,s')$.

So what? The genius is: given *s* and *a*, it is ***conditionally independent*** of all previous states and actions—i.e., the state transitions of an MDP meet the Markov property.

We want to maximize the total reward by assigning the best possible action to each state! Value iteration is the most popular algorithm to do so, to find control policies. It recursively calculates the utility of each action (value) relative to a reward function.

$$Q^{\pi^*}(s,a) = r(s,a,s') + \beta \sum_{s\prime} p(s,a,s')\, V^{\pi^*}(s')$$

where: $r(s,a,s')$ is reward, β is a discount factor, $V^{\pi^*}(s)$ is the value of the best policy π^* for state s, $Q^{\pi^*}(s,a)$ is the Q-value (utility) of the best policy for the state-action pair of (s,a), and finally:

$$V^{\pi^*}(s) = \max_a Q^{\pi^*}(s,a) \quad (\textbf{\textit{Note}}\text{: value function!})$$

Now, here is the value iteration algorithm:

$$Q_{t+1}(s,a) \leftarrow r(s,a,s') + \beta \sum_{\hat{s}} p(s,a,s')\, V_t(s')$$

$$V_t(s) \leftarrow \max_a Q_t(s,a)$$

Notes: The future term is the *expected cumulative reward* of state *s*. (Think about asynchronous dynamic programming in path-finding!) In a multiagent MDP, any (global) action *a* is really a vector of local actions $(a_1, \dots, a_n)$, one by each of *n* agents. In a way, it is variable elimination in the particular context of multiagent MDPs.

2.3. Negotiation, auctions, and optimization

A global problem is decomposed into subtasks, and distributed among a set of agents. Each agent has different capabilities. For each agent *i,* there is a function c_i such that for any set of tasks *T*, $c_i(T)$ is the cost for the agent to achieve all the tasks in *T*. The agents then enter into a negotiation process which improves on the assignment, and hopefully, culminates in an optimal assignment, that is, one with minimal cost. Furthermore, the process can have a so-called *anytime property*; even if it is interrupted prior to achieving optimality, it can achieve significant improvements over the initial allocation.

- Contract net protocol: contract host and bidders, auctions (Chapter 11)
- Direct 1-to-N, or multiple 1-to-1 negotiations (Advanced topics, if time permits)

At this point several questions may naturally occur to the reader.

- We start with some global problem to be solved, but then speak about minimizing the total cost to the agents. What is the connection between the two? (*Think about autonomy and emergent behavior!*)
- When exactly do agents make offers, and what is the precise method by which the contracts are decided on? (*Think about utility, future and current rewards, reinforcement learning!*)
- Since we are in a cooperative setting, why does it matter whether agents "lose money" or not on a given contract? (*Think about incomplete and dynamic environmental properties and optimality!*)

Now, consider the assignment problem. A (symmetric) assignment problem consists of:

- A set *N* of *n* agents;
- A set *X* of *n* objects;
- A set $M \subseteq N \times X$ of possible assignment pairs; and
- A function $v : M \rightarrow \mathbb{R}$ giving the value of each assignment pair.

Now consider that an assignment is a set of pairs $S \subseteq M$ such that each agent $i \in N$ and each object $j \in X$ is in **at most one pair** in *S*. A feasible assignment *S* is optimal if it maximizes $\sum_{(i,j)\in S} v(i,j)$.

Imagine that each of the objects in *X* has an associated price; the price vector is $p = (p_1, \dots, p_n)$, where p_j is the price of object *j*. Given an assignment $S \subseteq M$ and a price vector *p*, define the "utility" for an assignment *j* to agent *i* as $u(i,j) = v(i,j) - p_j$. An assignment and a set of prices are in *competitive equilibrium* when **each agent is assigned the object that maximizes his or her utility given the current prices**.

Definition 2.3.4. (Competitive Equilibrium). A feasible assignment *S* and a price vector *p* are in competitive equilibrium when for every pairing $(i,j) \in S$ it is the case that $\forall k\ u(i,j) \geq u(i,k)$.

Naive Auction Algorithm
$S \leftarrow \emptyset$ // Initialization
forall $j \in X$ **do**
    $p_j \leftarrow 0$ // initialization
end for
repeat
    // Bidding Step:
    let $i \in N$ be an unassigned agent
    // Find an object $j \in X$ that offers *i* maximal value at current prices:
    $j \in \arg\max_{k|(i,k)\in M}(v(i,k) - p_k)$
    // Compute *i*'s bid increment for *j*:
    $b_i \leftarrow \left(v(i,j) - p_j\right) - \max_{k|(i,k)\in M;k\neq j}(v(i,k) - p_k)$
    // which is the difference between the value to *i* of the best and second-best objects at
    // current prices (note that *i*'s bid will be the current price plus this bid increment).
    // Assignment Step:
    add the pair (i,j) to the assignment *S*
    if *there is another pair* (i',j) **then**
        remove it from the assignment *S*
    end if
    increase the price p_j by the increment b_i
until *S is feasible* // that is, it contains an assignment for all $i \in N$
End Algorithm

The problem, though, is that the above algorithm may not terminate. *This can occur when more than one object offers maximal value for a given agent; in this case the agent's bid increment will be zero. If these two items also happen to be the best items for another agent, they will enter into an infinite bidding war in which the price never rises.*

A terminating auction algorithm. To remedy the flaw exposed previously, we must ensure that prices continue to increase when objects are contested by a group of agents. The extension is quite straightforward: we add a small amount to the bidding increment.

$$b_i \leftarrow u(i,j) - \max_{k|(i,k)\in M;k\neq j} u(i,k) + \epsilon$$

Because the prices must increase y at least ϵ at every round, the competitive equilibrium property is no longer preserved over the iteration. Agents may "overbid" on some objects! Thus:

Definition 2.3.7 (ϵ -Competitive Equilibrium). A feasible assignment *S* and a price vector *p* are in competitive equilibrium when for every pairing $(i,j) \in S$ it is the case that $\forall k\ u(i,j) + \epsilon \geq u(i,k)$.

In other words, no agent can profit more than ϵ by bidding for an object other than his assigned one, given current prices.

2.4. Social Laws and Conventions

Consider the task of a city transportation official who wishes to optimize traffic flow in the city. While he or she cannot redesign cars or create new roads, he or she can impose *traffic rules*. A traffic rule is a form of a *social law*: a restriction on the given strategies of the agents. A typical traffic rule prohibits people from driving on the left side of the road or through red lights. For a given agent, a **social law presents a tradeoff; it suffers from loss of freedom**, but can benefit from the fact that others lose some freedom. **A good social law is designed to benefit *all* agents.**

In general, agents are free to choose their own strategies, which they will do based on their guesses about the strategies of other agents. Sometimes the interests of the agents are at odds with each other, but sometimes they are not. In the extreme case the interests are perfectly aligned, and the only problem is that of coordination among the agents. **Again, traffic presents the perfect example; agents are equally happy driving on the left or on the right, provided everyone does the same**.

A social law simply eliminates from a given game certain strategies for each of the agents, and thus induces a *subgame*. **When the subgame consists of a *single* strategy for each agent, we call it a *social convention***.

This leaves the question of how one might find such a good social law or social convention:

- Democratic perspective: how conventions can emerge dynamically as a result of a learning process within the population *(**Note**: Learning and Teaching and Voting!)*
- Autocratic perspective: imagine a social planner imposing a good social law (or even a single convention). The question is how such a benign dictator arrives at such a good social law. (***Note**: Mechanism design!)*

In general the problem is hard; specifically, when formally defined, the general problem of finding a good social law (under an appropriate notion of "good") can be shown to be NP-hard.